REALIZZAZIONE DI UNA SOLUZIONE IAC(INFRASTRUCTURE AS CODE) CHE CONSENTA IL RILASCIO DI UN'INFRASTRUTTURA PER AMBITI DEVOPS

Roberto Antoniello – 875693

Data di termine dello stage: 3 Marzo 2023

1 Ente presso cui è stato svolto il lavoro di stage

Il lavoro di stage è stato svolto presso Certimeter s.r.l., un'azienda nata come spin-off dell'Università degli studi di Torino con sede a Torino e Milano che tratta consulenza in diversi ambiti IT.

2 Contesto iniziale

Il contesto iniziale dello stage riguardava l'entrare in contatto con una nuova metodologia sconosciuta prima di questo tirocinio quale è il DevOps. Allo stato attuale è un metodo di sviluppo ancora in evoluzione, permettendo di sviluppare e rilasciare software molto più velocemente che in passato in maniera continua. Questo stage si è concentrato nello studio del DevOps e della sua applicazione simulando una casistica reale.

3 Obiettivi del lavoro

Gli obiettivi formativi prefissati per questo progetto di tirocinio sono stati i seguenti:

1. Acquisire competenze in ambito DevOps.
2. Acquisire conoscenze sul ciclo di vita di un'infrastruttura.

A livello tecnico l'obiettivo principale è stato quello di costruire un'infrastruttura in grado di essere rilasciata in automatico attraverso il cloud e sulla quale fosse possibile rilasciare applicazioni basate su microservizi. Per fare ciò sono state sfruttate le conoscenze apprese durante il conseguimento degli obiettivi formativi, con l'utilizzo attivo delle tecnologie studiate alle quali si è dedicato il capitolo successivo.

4 Descrizione lavoro svolto

Le principali fasi svolte durante il progetto di tirocinio sono state le seguenti:

1. Studio delle tecnologie coinvolte.

Durante questa fase sono state approcciate diverse tecnologie per la prima volta. Inizialmente sono stati appresi i concetti a livello teorico che ruotavano attorno al funzionamento di questi strumenti. Successivamente questi concetti sono stati messi in pratica applicandoli a diversi test, partendo da basso livello e salendo man mano di astrazione.

2. Progettazione dell’infrastruttura.

In questa fase è stata eseguita un’analisi dei requisiti e definita la struttura dell’infrastruttura finale. Per fare ciò è stata sfruttata la fase iniziale di studio per capire in che modo le varie tecnologie andassero collegate tra loro correttamente.

3. Implementazione del sistema.

In questa fase finale è stata implementata l’infrastruttura in funzione dei disegni progettuali prodotti in precedenza.
Infine è stata poi rilasciata al suo interno un’applicazione a microservizi. Come da requisiti iniziali anche le successive modifiche all’applicazione potevano avvenire in automatico in regola con la CI/CD.

5 Tecnologie coinvolte

Git: gestione e versionamento del codice.

Docker: costruzioni dei container.

Kubernetes: orchestrazione dei container.

Azure: cloud provider scelto per sfruttare il cloud computing.

Terraform: definizione dell’infrastruttura tramite stesura di codice.

Jenkins: tecnologia a supporto dello sviluppo di pipeline per il continuo rilascio automatico delle modifiche.

6 Competenze e risultati raggiunti

Al termine di questo progetto i risultati raggiunti sono stati diversi, li ritroviamo qui di seguito:

• *Appreso competenze in ambito DevOps.*

Nel corso del tirocinio è stato possibile approcciarsi da zero con questa metodologia di sviluppo, riuscendo a carpirne i vantaggi del suo utilizzo e a sviluppare nuove competenze.

• *Appreso conoscenze su diverse tecnologie.*

Grazie a questo progetto è stato possibile conoscere e acquisire competenze su molte tecnologie in breve tempo. In aggiunta al background accademico, tutto ciò ha arricchito molto il bagaglio di competenze informatiche.

• *Realizzata un’infrastruttura fedele ai requisiti iniziali.*

Terminato il progetto, è stata realizzata un’infrastruttura che rispetta i requisiti e i vincoli imposti nel corso del tirocinio, seppur con piccole modifiche rispetto a quanto fatto in sede di progettazione. Come accennato e approfondito al termine del capitolo 4, durante la fase di implementazione è stato reso necessario il non utilizzo di Nexus per il repository Docker

privato a causa di ostacoli tecnici deviando quindi su un normale Docker Hub privato per non sforare con i tempi.

Di seguito sono descritti i problemi incontrati durante il lavoro, distinti per ogni fase svolta, e gli approcci utilizzati al fine di risolverli.

- *Fase di studio*.

Nella fase di studio il problema maggiore è stato comprendere il ruolo di ogni strumento nel contesto specifico in cui dovevano essere utilizzati. Questo ha fatto si che la fase iniziale di studio sia stata lenta per poi gradualmente accelerare verso la sua fine.
L'approccio standard numero uno è stato leggere buona parte della documentazione ufficiale per capire i funzionamenti, eventualmente integrando queste informazioni sfruttando risorse sul web come ad esempio video molto approfonditi su YouTube. Man mano che le tecnologie viste e i concetti aumentavano, il modo più efficiente per cercare di tenere il tutto coeso ed evitare confusione è stato continuare a farsi determinate domande come ad esempio:
Se con Docker posso creare e gestire container, ma con Kubernetes posso fare lo stesso, qual è la differenza fondamentale? Ora che so che su Azure c'è un servizio che mi garantisce un cluster Kubernetes pronto e operativo, come posso fare per definirlo, modificarlo o cancellarlo in automatico rispettando i requisiti?

• *Fase di progettazione.*

In questa fase l'unica difficoltà è stata definire accuratamente i casi d'uso da implementare successivamente ed eventualmente escluderne altri. L'approccio è stato intuitivamente andare ad analizzare i requisiti iniziali e da essi esportare i possibili casi d'uso che potevano verificarsi sul sistema. Nel resto della progettazione non vi sono state ulteriori criticità considerando la chiarezza dei requisiti ed essendo appena usciti dalla fase di studio con tutti i concetti carpiti.

• *Fase di implementazione.*

Inizialmente, vi è stata una scelta di implementazione per quanto riguarda il tipo di oggetto con cui sviluppare le molteplici pipeline.
In Jenkins ci sono diversi modi per lavorare con le pipeline e un primo approccio utilizzato durante la fase di studio per poi testarlo in sede di implementazione è stato l'uso di uno script adibito a pipeline. Questo metodo di sviluppo è molto potente siccome è possibile definire tutti i vari blocchi in uno script unico all'interno di un file che verrà chiamato Jenkinsfile. Tuttavia, dovendo simulare una situazione reale, risulta difficile da mantenere all'interno di un team più ampio, avendo più utenti che ci devono mettere le mani sopra.
Dunque a risoluzione di questo problema è stato optato invece l'utilizzo di diversi progetti detti freestyle, ovvero dei job singoli ognuno con il proprio contenuti di azioni, allegando poi dei trigger di post-compilazione per regolare quale evento succede l'altro.
Durante la fase finale di rilascio, una delle difficoltà è stata il dover prendere in carico il

codice di un’applicazione scritto da un’altra persona. In un primo momento c’è stata la nacessità quindi di comunicazione per avere una panoramica sulla struttura architetturale dell’applicazione, così da avere un’idea più precisa di come i microservizi comunicassero tra di loro a livello tecnico. In un secondo momento invece è stata approfondita in autonomia la struttura osservando i vari repository per comprendere il punto esatto in cui si rendeva necessaria la modifica delle rotte e i relativi puntamenti.
Pur non avendo conoscenze approfondite degli strumenti di utilizzati dallo sviluppatore, in seguito all’analisi del sorgente è stato possibile ricavare dove effettuare le modifiche, adattandole poi con le diciture definite nei manifesti YAML scritti per Kubernetes. Un ulteriore problema riscontrato è stato quello di combinare la tecnologia Nexus con il cluster Kubernetes. Qui affrontiamo direttamente lo svantaggio dell’utilizzo di un servizio preconfezionato quale è AKS(Azure Kubernetes Service).
La criticità principale risiedeva nel fatto che il pull delle immagini da parte del cluster avveniva obbligatoriamente attraverso il protocollo HTTPS, mentre il repository ospitato su Nexus come opzione di default viaggiava in HTTP.
Un primo approccio è stato quello di provare a forzare l’AKS a eseguire il pull mediante registro non sicuro, ma è stato abbastanza chiaro capire che questo non era possibile.
Un secondo approccio è stato creare un certificato self-signed da applicare sul repository Docker così che fosse raggiungibile via HTTPS.
Nonostante ciò, per come era preconfigurato il cluster, non c’era modo di farsi dare la fiducia su quel certificato al container runtime in esecuzione sotto AKS.
Possiamo dire che i vantaggi di un servizio di questo genere sono andati a discapito di un eventuale bisogno di dover modificare la configurazione delle macchine. Ad esempio su un cluster Kubernetes nativo, una semplice soluzione poteva essere andare a modificare la configurazione di ogni nodo in modo che tutti si fidassero di tale certificato. Una volta aggirato l’ostacolo mediante l’uso di un normalissimo repository su Docker Hub, il resto dell’implementazione del sistema non ha avuto altre criticità rilevanti.

7 Bibliografia

- TechTarget, Demystify the DevOps process, step by step,2023. https://www.techtarget.com/searchitoperations/tip/Demystify-the-DevOps-process-step-by-step
- Ionos, I vantaggi del cloud computing. 2023 https://www.ionos.it/digitalguide/server/know-how/i-vantaggi-del-cloud-computing/
- Digitalocean, Infrastructure as Code Explained, 2020. https://www.digitalocean.com/community/conceptual-articles/infrastructure-as-code-explained